laval Research Laboratory
'ashington, DC 20375-5000

NRL Memorandum Report 6705

Battle Engagement Area Simulator/Tracker

R. L. KOLBE,* J. P. BORIS AND J. M. PICONE

Laboratory for Computational Physics and Fluid Dynamics

**Berkeley Research Associates*
5532 Hempstead Way
Springfield, VA 22151

October 8, 1990

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	1990 October 8	

4. TITLE AND SUBTITLE

Battle Engagement Area Simulator/Tracker

5. FUNDING NUMBERS

PE - 63223C
PR - DN159-068
WU - 44-3287-00

6. AUTHOR(S)

R. L. Kolbe,* J. P. Boris and J. M. Picone

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
Washington, DC 20375-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL Memorandum
Report 6705

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
800 N. Quincy St. BCT #1
Arlington, VA 22217-5000

SDIO
800 N. Quency St.
Arlington, VA
22217-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

*Berkeley Research Associates, 5532 Hempstead Way, Springfield, VA 22151

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Surveillance, Correlation, and Tracking problem is the computation-limited kernel of future battle management systems being developed for the military. In this report, we show how high-performance tracking can be performed on the Connection Machine (CM) using a Monotonic Lagrangian Grid (MLG). This high-performance algorithm scales better than traditional algorithms which scale as $N^2_{\sqcup}$ or $N^3_{\sqcup}$, where N is the number of objects. The CM, a fine grain Single Instruction Multiple Data (SIMD) computer, is described; and its affects on algorithm design are discussed. Fully parallel algorithms developed at the NRL Laboratory for Computational Physics and Fluid Dynamics for the CM are described for scenario generation, near-neighbor search within the MLG, sensor report generation, fusion of sensor reports and a tracking. Finally, a description of a Strategic Defense Initiative (SDI) operational system using a heterogeneous assembly of computers is given.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Battle Engagement Simulator/Tracker (BEAST)	50
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std 239-18
298-102

CONTENTS

BATTLE ENGAGEMENT AREA SIMULATOR/TRACKER

1 Introduction

The Battle Engagement Area Simulator/Tracker (BEAST) project was first proposed by Boris, et al. [1,2,3] to demonstrate algorithms and a problem solving framework suited to processing the sensor contact reports and tracks from tens of thousands to hundreds of thousands of objects on ballistic trajectories and in low earth orbit. The goal was to rethink such a system from the ground up with mathematically optimal or near optimal parallel algorithms scaling at least as lnN on an N processor system. Current correlation and tracking schemes generally have portions of scalar code and algorithm scaling as the number of objects squared in some of the compute-intensive portions. Presently, these simulators with correlation and tracking schemes meet real-time requirements for scenarios containing only a few hundred objects. Thus, realistic scenarios with even a few thousand objects generally have required an hour or more of computer time to simulate a few minutes of real time.

BEAST is a high-speed model for a simulation, correlation and tracking, and engagement program using a fully parallel Monotonic Lagrangian Grid (MLG) data structure, developed by the Laboratory for Computational Physics and Fluid Dynamics (LCP&FD) scientists, [4,5,6,7,8], to permit the desired lnN scaling throughout.

Manuscript approved June 23, 1990.

Though currently implemented as a demonstration on the 16,384 node Thinking Machines Corporation's Connection Machine* (CM) at the Naval Research Laboratory, additional degrees of parallelism can be tapped in an operational configuration. This could be accomplished, for example, through a heterogeneous element supercomputer system combining SIMD, MIMD and high-performance scalar/vector components where they are best suited. The project's purposes are to successfully apply parallel and supercomputer technology to the Strategic Defense Initiative Organization (SDIO) battle management tracking problems, and to develop software and algorithms appropriate for new, highly parallel supercomputer architectures.

The preliminary implementation of BEAST performs five "operations":

1. Model conflict scenarios (ground truth) consisting of tens of thousands of targets, decoys, sensors, weapon satellites, and other objects in orbit or on ballistic trajectories,

2. Simulate in a single system the sampling of ground truth by many loosely coupled sensors to produce contact reports from all sensors in a parallel fashion,

3. Construct and maintain in real time the necessary composite data structures containing reports from multiple sensors and tracks as the sensor reports are generated,

4. Process and analyze the data, producing projections of tracks tagged with estimated target identities and generating new tracks automatically,

5. Model interactions and decisions by battle management analysis systems using these parallel data bases constructed in MLG form.

The last item in the list includes many important aspects of the SDIO battle management problem which the BEAST project is only treating in a cursory way. These parts are evaluating tactical threats to defensive sensors, communication and weapon

*Connection Machine is a trademark of the Thinking Machines Corporation.

platform satellites, weapons allocation against detected targets, kill assessments, strategic situation assessment, and target discrimination. Many conventional simulations, sponsored by the SDIO are aimed at achieving these tasks. The BEAST project demonstrates a data-processing framework within which the projection of satellite orbits and target flight paths over appreciable fractions of an hour can be accomplished faster than real time. Then the subsequent "managerial" components of the problem will have suitable geotemporally organized data to work on and time in which to do this work. Using linked MLG data structures executing in an SIMD or MIMD parallel environment provides one convenient way to make the track and sensor report data bases available to the decision-making portions of a complete battle manager.

Each parallel MLG data structure contains a fixed size data set, which has been organized into a grid structure. Information flows into and out of this structured data set and is rearranged when information is added. This MLG data structure will be referred to as a "tank" since it has a three-dimensional structure similar to a rectangular tank of water [1]. Previously mentioned operations 1 and 2 of BEAST involve the ground truth tank (typically a $32\times32\times32$ MLG), the sensor subtank (a $4\times4\times4\times2^m$ MLG, comprised of the 64 sensor locations; see Section 5), and the multiple sensor report tank (typically a $32 \times 32 \times 32$ MLG which is presently 5 time levels deep). Operations 3 and 4 of BEAST involve the multiple sensor report tank and a tracking-correlation tank (typically, a $32 \times 32 \times 32$ MLG). Massively parallel sensor report generation, sensor control, and multi-sensor fusion algorithms have been constructed and a preliminary report-track correlation algorithm has been demonstrated which automatically generates new tracks. The ground truth scenario implemented in the BEAST consists of satellites, missiles and other objects (earth and hole nodes; see Section 3) totaling either a maximum of 4K (4,096) or 32K objects depending on the fraction of the computer system used and the size of the scenario. The smaller 4K system allows faster runs, typically a couple of minutes for a half hour of real time, and simplifies debugging. The larger 32K system is designed for denser, more realistic scenarios while still running somewhat faster than real time.

Figure 1 shows a flow diagram of the BEAST code. The blocks represent the functional sections ("functions") of the code. The block labels beginning with the word

Initialize perform the housekeeping duties for the code. *Initialize Program Control* reads the input data telling how many of what objects are to be considered, the types of output, orbital information, etc. *Initialize Scenario* uses the input data to create the physical starting conditions for the simulation. This function assigns sensor orbits, the type and number of defensive (blue) and offensive (red) objects, starting locations of missiles and satellites, etc. *Initialize Data Sets* creates the necessary MLG data structures for the remainder of the code. After the data sets have been generated, they are sorted for MLG order.

The code marches forward in time by continually repeating a sequence of tasks and functions. The functions shown within this loop are not necessarily performed every time step but only as needed. The *Update Scenario* function (operation 1) steps the objects forward in time producing new positions and velocities, evolving the scenario. The *Near Neighbor Search* is a gating algorithm to identify defensive and offensive objects within an critical distance of each other. This information could be used for weapons allocation. In a skeletal way, this function, operation 5 of BEAST, is the beginning of battle management analysis. It is presently used for graphical output.

The *Sensor Control* function assigns the scanning volumes to sensors and assigns storage regions for the sensor data within the CM. The allocation of storage regions depends on the sensor's position. Having assigned scanning volumes to the sensors, the contact reports for individual sensors can be generated. This task is performed in the *Multisensor Contact Report Generator.* These two functions, *Sensor Control* and *Multisensor Contact Report Generator*, are operation 2 of BEAST, multiple sensor report generation.

The multiple sensor contact reports are then fused in the *Multisensor Contact Report Fusion* function. This function combines information about a single object from all sensors into a single "track segment" and eliminates most duplicate contact reports. The track segments for each scan are correlated by the *Global Tracking and Correlation* function. This function connects the tracks, based on past track segments, to the new track segments and generates a new track for any track segment not associated with a track. These two functions, *Multisensor Contact Report Fusion* and *Global Tracking and*

Correlation, are operations 3 and 4 of BEAST. A follow-on report, based on a previous report which reviewed algorithms [9], contains a discussion of factors, assumptions and figures of merit presently used by BEAST.

The present report is intended as a major progress report of a project started originally under DARPA/ISTO sponsorship and now continuing under direct SDIO support. Though online, interactive demonstrations can be given and real-time videotapes constructed of complex scenarios, much work remains to be done. What has been accomplished to date is described in the next sections. Section 3 describes the parallel ground truth scenario generator on which the rest of the computations are based. Section 4 describes the MLG near-neighbors data structure in the context of the BEAST. Section 5 describes how the array of sensors are modeled in parallel on the CM. Section 6 continues this discussion with the parallel algorithms which fuse the distinct sensor reports into a composite data base for correlation with existing tracks. Section 7 describes the particular choice of parallel track-report association algorithms. Section 8 then considers a crude model for weapons allocation as an application of the dynamic MLG data structures automatically available as a result of the BEAST solution process. Section 9 describes some of the graphics and diagnostics currently employed in BEAST. Section 10 discusses some of the ways that an operational system would differ from the preliminary implementation of BEAST and what parallel computer architectures might be best suited to different components of the data reduction. Section 11 summarizes the results and discusses future developments.

2 The Connection Machine

Hardware

The CM is a front-end scalar computer (host) that controls up to 65,536 processors operating in parallel [10]. Each processor is smaller and less powerful than a personal computer but, under the control of the host, can store and manipulate distinct data sets (object, contact report or track) and perform the associated calculations. In this way the CM is well suited to problems like the SDI exoatmospheric battle management

problem where a number of similar autonomous events are occurring simultaneously with strong local relationships but weak global relationships. The CM, unlike other parallel processing systems, was particularly designed for fast execution of small data packet interchanges between arbitrary processors. This feature greatly simplifies the programming of parallel correlation and tracking algorithms and is ideally suited to the MLG restructuring algorithms needed for BEAST.

The program instruction sequence is stored on the host and each instruction is broadcast simultaneously to all the processors. Thus redundant storage and execution hardware for multiple copies of the program are eliminated. Based on a local context flag computed in parallel in each processor, that processor will perform or ignore the latest instruction. Thus complicated logic is easy to implement. BEAST is currently written in a parallel extension of the C programming language called C* [11].

A single chip within the CM contains 16 physical processors and is connected by Boolean n-cube wiring for interprocessor communications. Because of the wiring, the CM allocates 2^m processors ($m \geq 12$) which are arranged in an n-dimensional grid requiring each grid dimension to be a power of 2. Any program not using a power of 2 structure will waste resources, since the CM will allocate excess processors and these excess processors will not be used. Future CM architectures may remove this requirement. For the BEAST, a three-dimensional grid was chosen with equal lengths along the axes, either a 16 x 16 x 16, "small MLG data structure" or a 32 x 32 x 32, "medium MLG data structure", and in the future a 64 x 64 x 64, "large MLG data structure". These data structures result in scenario sizes of 4,096 (4K); 32,768 (32K) and 262,144 (256K) objects. This three-dimensional grid forms the base for the "functions" and MLG data structures of BEAST. An additional second grid is required for sensor control. It is a four-dimensional grid that is $4 \times 4 \times 4 \times 2^m$ where m is an integer such that 2^m equals the scenario size divided by 64.

The NRL has two CMs, each having two sections, and two host computers, a VAX and Sun. One CM has 4K physical processors per section and the other has 8K physical processors per section. The sections can be run either independently or together. To run a program, a user, who is logged in on a host computer, requests a

particular CM and the required number of sections. An additional CM feature is its use of virtual processors. The physical processors can have their memory divided into regions and thereby act logically as several distinct processors. Thus, the "medium" MLG data structure (32K) is run with either two virtual processors in each of 16K physical processors or four virtual processors in each of 8K physical processors. The "small" MLG data structure is run in 4K physical processors. Because of the limitations previously mentioned, the division of each physical processor into virtual processors will always be a power of two. The effects of this "virtual processor" subdivision on execution time is shown in Appendix A, which contains the results of timing tests. Throughout this report, "processors" will refer to virtual processors.

The CM has two additional hardware features. For data storage it has a data-vault, and for graphics it has a framebuffer [12,13]. The datavault consists of a file server computer, datavault disk controller and 42 separate disk drives. The datavault is connected to a CM I/O (input/output) Controller by a 64-bit I/O bus. A 64-bit data item is divided into two 32-bit sections and 7-bits are added to each section for error correction. The resulting 39-bit sections are stored on 39 individual disk drives with a single bit on each drive. The remaining 3 disk drives are spares. The datavault provides the high speed data transfer necessary to match the CM high computational capability.

A framebuffer resides in each of the CM's 4K or 8K physical processor sections. It is controlled by a sequencer, which also controls the processors and the CM I/O controller. The framebuffer and the CM I/O controller are connected to the physical processors by 256 I/O lines. The framebuffer contains a seven-megabyte display memory into which pixel values computed by the CM virtual processors are written and can be stored. The memory is arranged as 2048 x 1024 pixels, however only 1280 x 1024 pixels can be displayed on the monitor. Video can be displayed but the resolution is less than 640 x 512.

Communication

Communication between processors is performed by three methods:

1. Nearest-neighbor communication,

2. Remote-neighbor communication,

3. General communication.

For nearest-neighbor communication, each processor gets information from the processor next to it along an axis. For remote-neighbor communication, each processor gets information from the processor located the same specified distance and direction from it on the grid. For general communication, each processor gets information from an arbitrary distance and direction.

The CM can handle communication methods 1 and 2 very effectively because the processors are connected by Boolean n-cube wiring. A processor need not know the another processor's address. There is also no possibility of data collisions, that is, more than one message arriving at a processor. The processors on the grid are the nearest neighbors by communication links on the machine [14], and data transfer from one processor to the next along a grid axis is the fastest communication between processors. The CM handles item 3, general communication, using a "router". The router refers to the underlying packet-switching mechanism by which information is sent. The path chosen depends on the interprocessor distance, and the traffic on the system. By its nature, this form of communication will be the slowest on the CM.

The host computer can share data with the CM's processors in either of two ways. The first way is host array to processor transfer. In this form, an array on the host is dimensioned and sized as the n-dimensional grid of the processors. For BEAST, the host array would be $(16 \times 16 \times 16)$ for the "small" MLG data structure or $(32 \times 32 \times 32)$ for the "medium" MLG data structure. Element (i, j, k) of the host array would be transferred to a variable within the processor with grid indices (i, j, k). The reverse can also occur, that is the same variable within each processor can be transferred to a host array element having the same indices as the processor. The second way of sharing data is individual transfer between the host and a specific processor. The choice of method depends on the amount of data to be transferred.

3 Parallel Scenario Generation

The BEAST model needs a much faster than real-time (in our case a factor of 10 to 100) interactive driver to provide data on the evolving orbits and trajectories of all the objects in space. This allows most of the computer processing power to be available to the solver algorithms. We cannot pre-record the scenario as is done with much slower systems because simply reading in the data using conventional disk storage or tapes would be too slow, even with no processing by the host computer. The datavault could be used for storage; its use would limit the facilities available to run BEAST, and it may not have the necessary data transfer speed. Further, pre-recorded scenarios do not allow for interactive real-time modifications such as destroying a target or changing its orbit.

In the real world this "scenario generation" function occurs for "free" in front of the sensors. In a scaled up BEAST project, a separate computer would be used to simulate the scenario. We content ourselves with a small slice of the CM. We call this scenario the "ground truth" because, as far as the rest of the BEAST model is concerned, this simulation is reality.

In modeling the ground truth on the CM, each processor is assigned an item or a "hole" in the MLG data structure. A hole has a positional value in space and represents a location in the MLG data structure into which an object, report or track can subsequently be placed. The objects presently modeled are satellites in circular orbits and missiles in minimum energy trajectories outside the sensible atmosphere. The missiles are launched simultaneously from the center of two opposing rectangles in the northern hemisphere. The missiles are randomly assigned a target within two opposing rectangles. At present, the Earth is featureless, with a fixed radius and a simple inverse square force law to speed the scenario processing.

The initial positions and velocities are calculated for both satellites and missiles and the actual flight paths are advanced using a standard staggered leapfrog algorithm in the scenario update function. The algorithms are designed to guarantee non-decaying circular orbits and time reversibility of the integrations and are second-order accurate in

time. The current model is run with time slices of roughly ten seconds duration to match the processing rate of the sensor and tracker-correlator algorithms. This corresponds to about 1 kilometer accuracy or better in traversing 10,000 kilometers. Greater accuracy is possible but of course is beside the point since the scenario is the only ground truth the system can know. If the targets arrive on continuously accelerating hypergeometric orbits, the system should still be able to track them.

The organization of data describing these objects (and holes) on the CM is performed by an MLG. The MLG data structure built around the ground truth object locations will be called the TRUTH TANK. While the previous discussion does not justify the use of the MLG in this case (since the trajectory integrations themselves do not depend on whatever objects are nearby), future discussion will show the need for using the MLG data structure at this stage on the CM. The TRUTH TANK assigns each object in space to a particular processor based on the three-dimensional grid. This results in the placement of nearby objects in processors that are automatically near neighbors on the CM communication lines. A single processor will store the data for an object, sensor contact report, track segment, and track in distinct fields which are members of different MLG data structures.

The previously discussed scenario model, while not completely realistic, gives the object number and density of more complex models. To improve our model, the following items can be added:

1. multiple launch sites,
2. multiple warheads with decoys,
3. launch time window,
4. real scenario data,
5. loss and restoration of data signal
   (sensor satellites destroyed or communication links disrupted).

Items 1 and 3 will not noticeably change the program or timings. These items would be handled in the initialization section or by a flag for the processor. Item 2, multiple

warheads and decoys, could be modeled by associating an equivalent number of processors with a missile. A single processor would model the missile and based on release time or position, other processors, originally denoted as holes in the MLG, would be activated to represent a warhead or decoy. This action would be very similar to item 3. Item 4 will represent the greatest possible change to the present model and it will not be addressed in this report. The present model does account for missing or damaged sensors, which falls under item 5.

Appendix A contains the performance timings for the basic flight modeling and MLG sorting algorithms. It is apparent from the Table 2 that the ratio of virtual processors to physical processors is more important than the number of objects, which is equal to the number of virtual processors. The ratio of virtual processors to physical processors is called the VPR of the CM. The table shows the computer time for 4K objects on 4K physical processors, VPR = 1, and for 32K objects on 8K and 16K physical processors, VPR = 4 and 2. The computer time is proportional to the VPR. It appears from the table that if 32K physical processors were available for 32K objects, the run time would be nearly identical to the runs with 4K physical processors and 4K objects. More details about the run times are discussed in Appendix A. However, all performance times were less than simulated time.

4 Near Neighbors and the Monotonic Lagrangian Grid

The identification of near neighbors is a basic requirement of three BEAST operations: sensor reports, correlation and tracking, and battle management, (see Sec. 1). Near neighbors are objects or reports located in the same area of space and time and within a fixed distance. Information about positions must be shared to calculate the distance between data nodes, and this requires the transfer of information between processors.

The importance of the MLG becomes quite apparent from the previous discussion on CM communications (see Sec. 2). Since the MLG has placed the objects in a grid that has physically adjacent near neighbors in near-neighbor grid positions, only the near-neighbor processors must be checked. Secondly, because only near-neighbor processors

are checked, the communication can be performed along axes of the hypercube.

This near-neighbor component of the model was developed before the sensor report algorithms, because its methodology was needed in the sensor-report fusion. To find near neighbors, a comparison between objects is required. The processors contain only a single object and could request the information from the neighboring processors, but there is a better way. An object's position data and the related data required for testing and comparison are copied into another MLG data field. This data field called the "touring data" is transferred sequentially among the neighboring processors, gathering information and returning to the original processor. The sequence of nearest-neighbor communications along the axes used for this transfer is called a "tour." Appendix B has a detailed description of this operation. As the touring data passes through a processor, the near-neighbor distance to the locally stored object is calculated and checked. If the distance between objects is less than the specified distance and any additional criteria are satisfied, information is exchanged between the touring data and the processor's data set. The touring data finally is transferred back to the originating processor, and the data collected is transferred to the original data set.

Figure 2 shows a tour along the x-axis. The top squares are the processor's data set and are label by the processor's grid index. The bottom squares are the touring data fields and are labeled by their originating processor's grid index. Part A shows the original positions before the tour. Part B shows the positions during the tour, and part C shows the positions after the tour. The arrows indicate the direction of information exchange or flow. By design, the touring data field will visit processors with larger grid indices than the originating processor's indices, and so will tour only a subset of the near-neighbors. A correctly designed transfer path will result in all the near-neighbors being checked with a minimum of data transfers. The tour used to find the near-neighbors checks the grid neighbor one location on all sides of a processor. This tour results in a cube $3 \times 3 \times 3$ centered at the originating processor. BEAST also uses a tour checking a $5 \times 5 \times 5$ cube in fusing contact reports from multiple sensors. In applications for molecular dynamics, the tour has to check $5 \times 5 \times 5$ or $7 \times 7 \times 7$ to ensure that all molecules within the force range contribute to the calculation.

5 Sensor Modeling

Up to 64 sensors observe the space in user-specified Earth orbits. The sensors are assigned orbits from either polar or inclined orbital planes based on an Office of Technology Assessment Report [15]. For this phase, 54 sensors are assigned equally to 6 orbital planes. The 6 planes are equally spaced around the equator and have an inclination to the equator of 75 degrees. The remaining sensors are assigned to a polar orbital plane. The sensors within each plane are equally spaced at an altitude of 400 Km.

The sensor contact reports generated in the present implementation are based on "double flash" sensing which results in a provisional track segment. This reflects the supposed presence of independent input preprocessors in an operational system which generates accurate positions and approximate velocities of the objects. The position reported is the objects position as seen by the second flash. The velocity reported is based on the change in position between flashes, and the time between flashes. This technique is common in laboratory laser systems with thousands of particles embedded in a complex field.

Each sensor reports the objects found in an MLG index cube around it. There are two cube sizes: for the "small" MLG data structure the cube is $4 \times 4 \times 4$; and for the "medium" MLG data structure the cube is $8 \times 8 \times 8$. The location of the cube varies for each scan period and alternates the sensor's viewing volume. For the even sensor scans, the sensors look away from the Earth. For the odd sensor scans, the sensors look toward the Earth. This alternation in sensor scan volume results in almost complete coverage of all objects in space. Other algorithms are also possible for controlling the sensor scan volume but they have not yet been explored. The reports are then loaded, using the nested structure of a MLG, into a MLG data structure of the appropriate size, either "small" or "medium" MLG data structure. This data structure is called the REPORT TANK.

The CM's maximum speed is obtained by using all its processors to perform calculations. To generate sensor reports and use all the processors, each processor is assigned a different contact report and determines which sensor and object to use for

report generation. This procedure is required since some objects are scanned more than once and other objects are not scanned. Therefore, assigning an object's processor to generate the contact report would result in some processors generating more than one report and some processors generating no reports. Further, this would presuppose knowledge of the actual objects. If the sensor's processors were used, only 64 processors would have to generate the 4K or the 32K contact reports. For this parallel procedure, each processor calculates its sensor assignment. The processor requests information about that sensor and then calculates its object assignment. The processor requests information about that object and generates the contact report. If the processor is not associated with either a sensor or object, the processor generates a hole value for the REPORT TANK.

The report generation procedure utilizes the nested structure of the MLG, which will be explained in a follow-on report. Each sensor has a small cube, a MLG data structure whose size depends on the size of the REPORT TANK, associated with it in the REPORT TANK. The location of this small cube is controlled by a special MLG data structure called the SENSOR TANK. The 64 sensors and their small cubes require only a $4 \times 4 \times 4$ data structure for control, but there are 4K or 32K processors. The limits imposed by the CM architecture and C* result in a second processor gird and MLG data structure for the SENSOR TANK. The grid and data structure for the SENSOR TANK are $4 \times 4 \times 4 \times 2^m$, where m is an integer such that m equals the number of processors divided by 64. Presently, only the $4 \times 4 \times 4 \times 1$ hyperplane of the SENSOR TANK is used but future revision could use the remaining hyperplanes. The code switches between the two grids depending on the tank being accessed. The processors containing the sensors transfer positional and other related data about the sensors to the processors forming the first hyperplane. These data are used to arrange the SENSOR TANK in MLG order. If a sensor does not exist, the grid point within the SENSOR TANK is assigned a hole value. Appendix C discusses alternate methods and procedures for forming the SENSOR TANK and carrying out the data transfer. Each SENSOR TANK grid point represents a location of a cube within the REPORT TANK. The sensors and cubes are, thereby, associated by the sensor's location within the SENSOR TANK.

In the REPORT TANK every processor within the a given sensor's cube will

generate a different report for the sensor. The object reported by a processor will depend on the object's location with respect to the sensor, the processor's location within the cube and the scanning view being used. These reports within the cube are in MLG order and the cubes have been located within the REPORT TANK based on the order of the sensors in the SENSOR TANK.

Since a fixed number of objects is reported per sensor, the three-dimensional scanning volume will vary in spatial volume from sensor to sensor based on the density of objects around the sensor. A high object density around a sensor will quickly fill the MLG grid around the sensor within the TRUTH TANK with objects near in distance to the sensor. This results in the report quota being filled by only objects very near the sensor and in a small scanning volume. The opposite is true for a low object density around a sensor. With a low object density, we will fill the MLG grid around the sensor within the TRUTH TANK with objects further away in distance from the sensor. This results in a larger three-dimensional scanning volume.

While the modeling of the sensor does not appear very detailed, the effects are similar to more complex models, which have objects moving into and out of a sensor range, objects detected by more than one sensor, and objects undetected. There are five items that should be addressed by further study. These items are

1. range of sensors
2. reports without velocities,
3. errors in positions and velocities,
4. IR or two angle reports,
5. sensor resource management.

Item 3 can be easily added with each processor calculating a random addition to its contact report's position or velocity. This random addition would be chosen from a known or postulated distribution. Items 1 and 4 would require some new functions to be added to the model. Item 4 has been treated by MLG data structures in a previous report [16]. A second option may be to assign these functions to another computer,

which would process this data to produce the required reports. Item 2 can be added easily but the tracking and correlation algorithm would require modification.

Sensor resource management, item 5, includes multiple sensor fusion nodes, volume surveillance responsibility and data storage allocation. Passive sensors, e.g. IR sensors, will require coordination of platforms to obtain high quality track segments. Therefore, sensors must report matching space volumes. Also, the duration of the configuration must be estimated. The present BEAST project is beginning to develop the necessary algorithms now that a functioning framework has been demonstrated.

6 A Parallel Multi-Sensor Fusion Algorithm

Once the multiple sensor contact reports are loaded into the REPORT TANK, multiple contact reports of the same object at the same time can be identified and fused. The REPORT TANK must first be sorted to guarantee global MLG order with the positions of the objects mapped into a monotonically increasing index ordering in all three directions. The contact report initially assigned to a processor may not be in proper MLG order since its placement was based, in part, on the sensor location. The sensor control function must try to minimize the rearrangement required to establish global ordering. Minimizing the sorting will improve real-time preformance. After the sort, a tour is conducted on the REPORT TANK to eliminate any duplicated contact reports which lie with the tour boundaries. The tour checks MLG locations up to two grid indices away from each contact report. This tour checks a 5 x 5 x 5 cube centered at the contact report's processor, is similar to the near-neighbor tour, see Section 4. If two contact reports have positions within a fixed distance of each other and if their other attributes, such as velocity, are compatible, the contact reports are considered to be the same and one of them is eliminated from the REPORT TANK. Since a contact report of the same object by multiple sensors has roughly the same information, keeping multiple contact reports adds little to the model. However, not all duplicate contact reports are necessarily eliminated by the tour. These few remaining duplicate reports are addressed by the tracking-correlation algorithm. The contact reports remaining after fusion will be referred to as "track segments", since they become part of the track generated by

the tracker-correlator.

If the five items mentioned for further study in Section 5 result in changes to the sensor contact reports, then modification of the fusion module will be required. Several authors have independently proposed an algorithm for correlating observations of a number of objects by two or more passive sensors with overlapping fields or view [16,17,18]. Passive IR sensors will require platform configuration, volume assignments and data storage allocation. These functions would be performed in the sensor control section of BEAST (see Fig. 1). The resulting sensor focal plane data would necessarily require different sensor modeling and a different multi-sensor fusion algorithm. The multi-sensor fusion algorithm would combine focal plane tracks or reports to generate three-dimensional position and velocity track segment vectors which are more accurate than single sensor tracks. The resultant tracks could then be used by the present multi-sensor track fusion and global tracking and correlation code.

7 Parallel Tracking and Correlation Algorithm

The section of the BEAST program requiring the most development is the tracker-correlator (T-C) algorithm. A first cut parallel T-C module has been added to the model and is being tested. An object's track can be thought of as a string passing through an ordered sequence of beads, where the beads are the progressively newer track segments that belong to that track. The tracks connect the new track seqments to previously matched track segments. Using the track data previously assigned to the tracks, positions and velocities are projected forward. Because the velocities are estimated, an error results in the projected positions compared to the actual positions of the objects tracked. The tracks (processors) search a limited section of the REPORT TANK for matching track segments. The near-neighbor tour used for near neighbors and multi-sensor fusion cannot be used (at present) for this task. Since all the objects are not seen by the sensors in each scan period, a larger search pattern is required and potentially starts at a different offset in the REPORT TANK from the track's processor for each track. The tour does not allow for these differences unless it visits a very large number of processors. The method currently used for track/report correlation is one

of many ideas being studied. The graphics and diagnostics must be developed further before we have adequate online performance measurements of these T-C algorithms.

Based on some preliminary timing, this T-C section of BEAST doubles the execution time. For a simulation of 650 seconds, 65 time steps of 10 seconds each, on the "small" MLG data structure, the execution time without the T-C section was 20 seconds. The execution time with the new module and with the tracking performed every 6^{th} step was 46 seconds. Detailed discussion of the timing for this and other modules are given in Appendix A. The algorithm relies heavily on general routing for communications with most of the processors active. General communications are used because each processor is communicating with others in a random fashion. The distance and direction is different for each processor. From the previous discussion on the CM, (see Sec. 2) any reduction in this form of communication should increase the execution speed. This increase will be used to increase the frequency and of detail T-C calculations correspondingly.

Figure 3 shows the relationship of multiple sensor contact reports, track segments and tracks. Each sensor has a set of sensor contact reports associated with it. An object can have a contact report in multiple sensor's files. In this case, object a is reported by sensors 1 and 3, but not by sensor 2. These multiple contact reports are fused into the track segment. The track segment 12 for scan period t_p results from fusing contact reports for object a from sensors 1 and 3. The track segment file represents the contact report information from all sensors in this scan period. The track points to the track segments for the present and previous scan periods. Track A points to track segment 12 for scan period t_p and track segment 34 for scan period t_{p-1}. The track has no pointer to scan period t_{p-2}. The track represents the information from all sensors for all stored or previous scan periods.

One advantage for the T-C algorithm is that the tracks and track segments are in MLG order. The track MLG data structure is called the CORRELATION TANK. The MLG data structure has assigned tracks to processors and serves as an automatic gating algorithm between the track segments and tracks. The principle product of this gating is having the two data sets, REPORT TANK and CORRELATION TANK, in

processors having near-neighbor communication links. Not all the track segments will have tracks and not all tracks will have a track segment within these data structures. Therefore, the track must be projected into the REPORT TANK and matched to the track seqments. If any track segment is unclaimed, it starts a new track. If a track has not claimed any new track segments after a specified number of scans, it is eliminated from the file. Thus, track initiation and tree pruning are automatic.

Each processor containing a track locates a a track segment having a position near its track position and possibly located in another processor. Information is then requested about the processor containing this track segment. This operation is repeated for all nearby track seqments because more than one track segment may correspond to a track or the initial location process may not locate the best track segment for the track. It is also possible for more than one track to claim the same track segment, and it is possible that a track will not find and claim an associated track segment. At this point, comparisons are made to assign new track segments to tracks. In addition, comparisons are also made to fuse any duplicate track segments that have escaped the initial multi-sensor report fusion and to fuse any identified duplicate tracks. Any unclaimed track segments are assigned to new candidate tracks and the new data set sorted for MLG order.

Some track – track segment connections are not being made. This results in duplicate tracks being generated. These duplicate tracks are identified and removed by the method described during the comparison process. Both the track – track segment connections and duplicate track identification must be improved in effectiveness and speed. More details and future developments will be given in the next report.

8 Weapons Allocation: An Application

Allocating orbiting and available kinetic kill weapons to the incoming waves of targets in a manner which minimizes the likelihood of penetration is a major computational task with local and global implications which are both tactical and strategic in nature. This allocation cannot be performed meaningfully without a reliable tracker-correlator

operating to provide a useful approximation to ground truth, but it provides a good test of parallel techniques because significant portions of the overall problem can be defined.

Determining ahead of time which orbiting or ballistic objects of a particular class will pass within a specified distance of objects in another class later is a simplistic but relatively meaningful view of one kind of weapons allocation application. This task also happens to be useful as a prototype for other important adjacency computations required for controlling sensors, preprocessing contact reports, and processing track data. Thus, we chose this aspect of the problem in the BEAST project to focus development of the near-neighbors data structure (see Sec. 4) and the near-neighbor tour (see App. B). It was convenient to do this very early in the program development because only the time evolution of the TRUTH TANK is needed as an input data structure if we assume that the tracks simulated there have been generated, in fact, by a fully functional tracker-correlator algorithm rather than the ground truth simulation.

The calculation proceeds by tracing forward in time the evolution of all known tracks, asking which defensive (blue) weapons platforms pass within a prespecified gating distance of the nearby offensive (red) satellites and missiles. Graphically (see Fig. 4) these geotemporal conjunctions are colored gold as long as they last. Practically speaking, the gating distance could correspond, for example, to orbiting Kinetic Kill Vehicle buses which could put one or more KKVs directly on the target given a close enough conjunction and enough lead time. The greater the lead time, the smaller the onboard velocity change required to make any particular interception and the larger the spectrum of targeting options which exists.

Further, since many allocation options have to be evaluated, the faster the computation, the greater the level of overall optimization which can be considered. Actually, the fuel cost of some orbital adjustments is greater than others. Therefore, the gating distance which should be used would have to be a complicated function of details of the two orbits rather than the simple constant used in the BEAST tests here. The computational cost of these extensions would not be great, however, since most of the CM time goes into the distance gating computations and into restructuring to MLG. Using MLG indices as a measure of proximity is really a coarse-grained gating which prevents

large separations from even being checked. Section 10 discusses how similar weapons allocation computations would probably be conducted in an operational system.

9 Graphics and Diagnostics in BEAST

In the development of the model, several online graphics and diagnostic functions were produced in addition to the specific data printouts needed to test and debug portions of the code. The graphic displays use the online CM framebuffer directly or produce related numerical output files to be plotted using the interactive VOYEUR system developed in the LCP&FD. Four graphical displays are presently implemented:

1. the scenario,
2. detected objects,
3. sensors,
4. multiple sensor volumes.

The hardcopy of these displays was generated using the VOYEUR system at the LCP&FD but similar figures are generated by the framebuffer as the computation evolves.

The first display is the TRUTH TANK, plotted after the near-neighbor tour as described in Section 4. Figure 4 shows this graphical output, allowing the total picture to be seen. The second display is the REPORT TANK overlaid on the TRUTH TANK. Figure 5 shows this output, showing what objects were scanned this step or scan. The third display shows the sensor positions without the confusion of the other objects. The final graphic shows the objects sensed, each with a color based on the sensor which scans it, overlaid on ground truth with the actual objects shaded gray. This display allows the respective scanning volumes of sensors to be seen.

Currently, the full power of the framebuffer is not being utilized. The model uses only simple but relatively slow graphic functions. Future work will utilize more of the capability of the CM and the framebuffer by storing the background and redisplaying

it. Additional speed can be obtained by only plotting the image changes but such refinements were not necessary for the development of the basic BEAST scheme.

The diagnostic functions were originally developed to monitor the model and check results. The functions allow individual objects, classes of objects or sections of the MLG to be isolated in printed output. These requests for printout from selected processors must be done judiciously. If all processors are active, 4K or 32K lines can be printed.

10 Model Differences from a Full Paralleled Implementation - Recommendations

Using a single SIMD computer to perform all aspects of the BEAST computations has its advantages and drawbacks. The advantages are as follows:

1. A single, production-grade computer system can be used to test algorithms and data structure concepts.

2. Developmental communications between heterogeneous systems and operating systems do not need to be invoked - though tools like **Network Linda**, if uniformly implemented could be used to solve the logical problems of parallel process control.

3. Concentrating on the efficient SIMD implementation of all aspects of the problem provides, when complete, a limiting case of parallel performance with the fastest performance figures currently available. Related simultaneous efforts have been underway until recently on coarse-grained, Cray, and medium-grained parallel architectures to augment the experience gained on the Connection Machine.

4. For many of the tasks that need to be performed, efficient SIMD implementations are the most difficult to achieve while promising the greatest cost effectiveness if successful. For some compute-intensive tasks, efficient SIMD implementations may be the only cost effective way to accomplish the necessary data reductions.

The corresponding disadvantages of extending the current model directly to an operational system are as follows:

1. The Connection Machine's SIMD architecture is not optimal for all aspects of the problem. An operational system should employ different architectural components depending on the actual tasks to be performed.

2. In BEAST, the different tasks have to be executed sequentially on a single computer whereas in an operational system many of the tasks could all be operating simultaneously. Therefore, a greater degree of performance would be readily attainable through hierarchical parallelism than a single-system model such as BEAST can hope to obtain.

3. Fault tolerance is low in the CM model, as there is no automatic redundancy built in. This could be easily overcome in principle, for example, by using two autonomous sections to perform duplicate scenarios and then comparing results.

Therefore, the preliminary implementation of BEAST is an evolving benchmark demonstrating an efficient partitioning of the overall problem into bite-sized, fully parallel data structures and algorithms which are at least adequate to digest the problem. In this section, we want to sketch how the present BEAST might be extrapolated to an operational system. In the course of this discussion, it is appropriate to indicate what low risk computer architectures might actually be optimal for different aspects of the solution procedure in an operational heterogeneous system. This discussion could be very long and indeed might be viewed as one of the more important products of the program. Such a digression, however, would be premature at this stage, so the discussion will center on the kinds of considerations found to be important.

The scenario generator, which would have to be present even for an operational system (to test it), is not available in a real conflict. Thus, detailed knowledge of the ground truth is available in principle for only the defensive half of the scenario. An operational system would probably deal somewhat differently with this information than BEAST. Whereas BEAST ignores the TRUTH TANK information except to generate what would be the sensor reports, an operational system might well keep a defensive

TRUTH TANK, which could be used to remove contact reports clearly associated with our own hardware. This self-gating would have the effect of reducing significantly the computational burden of correlating the remaining contact reports with tracks.

In an operational system, such weapons allocation computations as we have discussed in Section 8 would be going on continuously and simultaneously in an independent auxiliary computer system dedicated to this application. For satellites with beam weapons the proximity considerations would be different from those with kinetic kill vehicles. For defenses based on a number of different systems, the weapons allocation process would be even more complex. While SIMD seems optimal for the coarse gating, it would indeed be surprising if the more complex, higher-level decisions were not made using an MIMD or even a fast scalar architecture.

Similar considerations apply to multi-sensor contact report fusion. The most difficult jobs encountered in BEAST relate to the construction of the sensor contact report MLG data structure and the sensor control proportions of the algorithm which determine where each sensor looks during the next scan. To make rapid progress, we have chosen to define identically shaped sub-MLG regions about each sensor which makes construction of a global MLG straightforward on the CM. Actually it would make more sense to have a medium grained Hypercube or Butterfly with one node per sensor carrying out these computations. The nodes would be assigned and updated to maintain MLG near-neighbors ordering of the sensors, but the increased flexibility of independent programming on each sensor node would facilitate different fields of view and the correlation of pairs of sensors to convert angle-only images to locations in three-dimensional space.

In such a problem partition, incoming sensor reports would be directed asynchronously to the nodes representing that sensor for preprocessing. This preprocessing would involve sorting all the contact reports from the sensor into a locally structured MLG which can subsequently be transferred to an SIMD data structure for global fusion. The same MIMD computer or even a powerful auxiliary scalar processor would be used to assign initial ranges of indices for these sub-MLGs in the global sensor report data structure.

In the track segment – track correlation stage of data reduction, an SIMD architecture running a tour algorithm like that in BEAST seems a good choice. Where an operational system would deviate is in using SIMD as a gating computation rather than the final assignment analyzer. For this task two other architectures would be more profitably employed. The gating would determine tens of thousands of contact reports which should be used to update particular tracks. The actual assignment computation using Kalman filtering should almost certainly be performed by a small, specially designed systolic array simply pouring out these assignment computations in pipeline fashion. There would also be many hundreds or even thousands of contact reports for which the correct assignment is not obvious but where additional, specialized computation by individual scalar processors (perhaps organized in MIMD fashion) could use additional information and techniques not available to the SIMD gating procedure which has to be applied to all the data. Such auxiliary computations could be expected to increase the current 90% capture rate per scan up to 98% or more.

Target discrimination and kill assessment are additional examples of compute-intensive algorithms which might better be executed by copying the updated track MLG data structure into an asynchronously operating MIMD data base. Rather more extended deliberations would then be possible while the major SIMD sensor-report fusion and report track correlation proceeds simultaneously.

Finally, the paradigm assumed here of centralized processing with all the contact reports and tracks being assembled and updated in one place is clearly possible. However, more likely is a distributed system of processing, with a global backup on the ground being used for developing the integrated strategy needed for long-range resource allocation and weapons deployment. Each orbital battle management platform could independently carry the computational power necessary to develop and maintain its local sensor and track data structures, perhaps using information from its own or nearby sensors. In this context the MLG data structure is particularly useful because it lends itself uniquely to building up global views from local nested or sub MLGs in building block fashion.

In this regard, our current ability to treat of order 32,000 objects in a quarter of a CM faster than real time means that a complete scenario with 1 million objects could be partitioned into 64 autonomous battle management computers orbiting in a $4 \times 4 \times 4$ dynamic real-world MLG. The CM is not flight worthy, but today's computer technology is sufficient to make one which is, if this were deemed necessary.

11 Summary

At the present time, BEAST is being implemented on the Thinking Machines Corporation's Connection Machine at NRL. The CM is comprised of a scalar host computer which broadcasts instructions to 2^n simple processors simultaneously. The NRL's CM can be user-configured at run time into 4K, 8K, or 16K processor systems. The instruction set broadcast by the host is performed by each processor on its data set. This allows each processor to handle a single object, sensor contact report or track. Since 2^n items are calculated in parallel, the total time required is the time for a single item to be calculated on one of the processors. One major consideration for the CM is ensuring use of all the processors for calculations. The CM's speed can only be obtained if all processors are used. A second major consideration for the CM is the communications between processors. Near-neighbor communication is the fastest and is maintained by MLG. Each MLG places near-neighbor objects, sensor contact reports, track segments or tracks into processors that are near-neighbors. This action also reduces general router communication. By using the fastest communication path and reducing the usage of the slower general router communication, sufficient speed can be maintained for real-time modeling.

The present model generates positional data for satellites and missiles. The satellites have circular orbits and the missiles have minimum energy flight paths outside the atmosphere. Sensor contact reports are generated by a "double flash" method. The sensor contact reports contain the position and an estimated velocity. These estimated velocities project some error into subsequent calculations, as would occur in a real system. The sensor reports those objects near it based on MLG indices. The volume scanned by each sensor is outward, away from the earth, and inward, toward the earth,

on alternate scans. The results of this scanning procedure is almost complete coverage of the objects in space. The sensor contact reports are then fused into track segments. After multi-sensor contact report fusion, the track segments are correlated with previous tracks. The model is undergoing tests for tracking and correlation. The run times for the model are less than real time. In future reports, we will describe in greater detail the factors, assumptions and figures of merit identified in the IDA report [9] and discuss these with respect to the current and evolving BEAST system.

Acknowledgement

The authors would like to thank Dr. Bruce Wald and Mr. Steve McBurnett of the U.S. Naval Research Laboratory for pointing out this important problem area to us and for their encouragement and support of the efforts in our laboratory. Discussions and work on related research topics with Miguel Zuniga, Felix Rosenthal, John Reynders, Jeff Uhlmann, Joe Collins, and Miles Redmon have played an important part in our being able to carry out this work. In particular, John Reynders and Jeff Uhlmann blazed our way in the use of the Thinking Machines Corporation's CM with the first cuts at some MLG restoration routines which evolved into components of the current BEAST project. Special thanks to Robert Whaley of Thinking Machines Corporation for his support and advice. Robert Whaley has written support programs which are components of the current BEAST project. The work reported here was supported by the SDIO under PMA 2300 Task 4 and was initiated under the DARPA/ISTO and DARPA/DSO projects.

References

[1] Boris, J.P., Picone, J.M. and Lambrakos, S.G., "Beast: A High-Performance Battle Engagement Area Simulator/Tracker", NRL Memorandum Report 5908 December 1986.

[2] Picone, J.M., Lambrakos, S.G., Boris, J.P., and Jajodia, S., "Initial Comparison of Monotonic Lagrangian Grid and Alternative Data Base Structures", NRL Memo-

randum Report 5860, September 1986.

[3] Jajodia, S., Meadows, C.A., Boris, J.P., Picone, J.M., and Lambrakos, S.G., "NRL Research in Database Management for the SDI Battle Management System", NRL Memorandum Report 5921, January 1987, originally prepared for Presentation & Proceedings, 1986 Military Communications Conference (MILCOM'86), Monterey, CA, October 1986.

[4] Boris, J.P., "A Vectorized Near Neighbors Algorithm of Order N Using a Monotonic Lagrangian Grid", J. of Comput. Phy., **66** (1), 1–20, 1986.

[5] Boris, J.P., Lambrakos, S.G., "A Near Neighbors Algorithm of Order N Using a Monotonic Logical Grid", First International Conference on Free-Lagrange Methods, Hilton Head Island, South Carolina, 4–6 March 1985, published in **Lecture Notes in Physics No. 238, The Free-Lagrange Methods**, Fritts, Crowley, Trease (eds), pg 158–181, (Springer-Verlag, New York, 1985).

[6] Boris, J.P., and Lambrakos, S.G., "Dynamical Organization of Evolving Data Using a Monotonic Logical Grid", Invited Paper & Proceedings, 1985 Summer Computer Simulation Conference, Chicago, Ill., 22-26 July 1985.

[7] Lambrakos, S.G., and Boris, J.P., "Geometric Properties of the Monotonic Lagrangian Grid Algorithm for Near Neighbor Calculations", J. of Comput, Phy., **73** (1), 183–202, 1987, also NRL Memorandum Report 5761, April 1986.

[8] Picone, J.M., Lambrakos, S.G., and Boris, J.P., "Timing Analysis of the Monotonic Logical Grid for Many-Body Dynamics", SIAM Journal on Scientific and Statistical Computing, 1987, **11** (2), 368–388, 1990.

[9] Frenkel, G., Paterson, Thomas S., Smith, Maile E., "SDI Battle Management/C3 Algorithms Technology Program Plan", Institute for Defense Analyses, April 1988.

[10] Hillis, W. Daniel, "The Connection Machine", Sci. Am. **256**, 108-115, 1987.

[11] *C* Reference Manual* Thinking Machines Corporation, Cambridge, Massachusetts, August 1987.

[12] *DataVault Reference Manual*, Thinking Machines Corporation, Cambridge, Massachusetts, February 1988.

[13] *Display Operations Reference Manual*, Thinking Machines Corporation, Cambridge, Massachusetts, August 1988.

[14] *Introduction to Programming in C/Paris*, Thinking Machines Corporation, Cambridge, Massachusetts, June 1989.

[15] U. S. Congress, Office of Technology Assessment, "Ballistic Missile Defense Technologies", OTA-ISC-254 (Washington, D.C. : U.S. Government Printing Office, September 1985).

[16] Picone, J.M., Boris, J.P., Lambrakos, S.G., Uhlmann, J., Zuniga, M., "Near-Neighbor Algorithms for Processing Bearing Data", NRL Memorandum Report 6456, May 1989.

[17] Black, T. R., *Journal of Guidance, Control, and Dynamics* **6** (1), 62 (1983).

[18] Varad, Rajan, "Scalar Correlation Algorithm Multi-Target Multiple Sensor Data Fusion", Present at the First National Symposium on Sensor Fusion, Orlando, FL, April 4-8, 1988.

Appendix A: Sample Timings

The world of large-scale computing has seen a number of major advances in the last ten years. Today's supercomputers, for example the Cray, the BBN Butterfly, and the Thinking Machines Corporation's CM, can perform calculations 100 times faster than last decade's computers. Tomorrow's computers promise the same increase in speed over today's. Not only has the speed increased, but the memory and hence the size of the problems that can be solved has also increased proportionally. The Butterfly and CM are two highly parallel computers employing a large number of distinct, rather inexpensive processors working together to solve a big problem. The Cray is a vector computer, getting its high performance from a few expensive, but very powerful processors. The Butterfly is an MIMD computer; its processors execute Multiple Instructions on Multiple

Data. While the CM is an SIMD computer, broadcasting to all the processors a Single Instruction which operates on Multiple Data. These supercomputers require software and special algorithms customized for their architectures to obtain performance near their theoretical limits.

In addition, heterogeneous supercomputer systems are being developed to allow simultaneous execution of several different computational tasks, giving higher performance by exploiting parallelism at several levels simultaneously. A heterogeneous system consists of several supercomputers with different architectures linked by a fast communication network. The system work load, often a single complex problem with many different parts, is divided among the computers, taking advantage of the particular computer architecture best suited to each part of the problem. The necessary software is difficult to write and maintain, since synchronization of computers and communications is a major problem. Software tools, such as **Linda**, a portable, high-level, parallel programming environment conceived at Yale University, are being developed to simplify the software development process and to organize and monitor the necessary interprocessor exchanges of information during computation.

In an actual battle, data describing approximately 10^4 to 10^6 objects will have to be analyzed by the battle management system. These objects include boosters, satellites, reentry vehicles, decoys, and debris. To develop and test the battle management codes, imaginary engagements called "scenarios" have to be constructed, providing input data to the battle manager programs as if a suite of orbiting and ground-based sensors were actually seeing an engagement develop in space. Such "scenario generators" calculate the locations, velocities, apparent brightnesses, and other related properties sensed for each of the objects. For passive sensors such as optical and infrared cameras, these properties include the target's temperature, cross-sectional area, and emissivity, but the range to the object can best be determined by correlating observations of two or more sensors. For active sensors such as radar and lidar, the field of view may be quite small, but the range is usually available directly. At the present time, most computers, VAX, SUN, Butterfly, and Cray XMP, cannot generate a scenario and perform tracking computations for a realistically large number of objects in real time. Table 1 shows the estimated execution time for various sized scenarios on different hardware

Program Run Time for One Hour of Real Time versus Scenario Size and Hardware Configuration

	100 Objects	10,000 Objects	1,000,000 Objects
VAX, SUN, ... :	$\approx$ 1 Hour	6 Weeks	3 Years
$\times 10$			
Butterfly, ...:	$\approx$ .1 Hour	$\approx$ 2 Days	3 Months
$\times 10$			
Cray XMP, ...:	$\approx$ .05 Hour	$<$ 1 Day	5-7 Days
$\times 10$			
TMC CM2 :	$<$ 1 Hour	$<$ 1 Hour	$<$ 1 Day
$\times 10$			
New Connection Machine	$<$ .1 Hour	$<$ .1 Hour	$\approx$ 1 Hour

Table 1: Estimated Run Times for Different Computers

configurations. It is for this reason that BEAST was developed.

As a first step in the development of BEAST, the CM SIMD architecture was chosen, and initial timings were performed. These timings used a simple model of circular orbiting satellites. The orbital position was based on the initial grid indices and the orbital velocity was randomly assigned. The satellite's flight was stepped forward in time with the positions, velocities and MLG data structure updated every step. The positions and velocities were updated using a staggered leapfrog algorithm.

The MLG data structure was updated using a checkerboard pattern sort, in which every processor is either even or odd based on its grid indices. For each axis, two comparisons, called a comparison set, are made. For the first comparison, processor pairs are formed along an axis by an even processor and the neighboring odd processor with the larger axis index; e.g., for the x-axis (i-index), the processors with $i = 4$ (even) and with $i = 5$ (odd) would form a pair. The axis coordinate values of the two objects within the processors would be compared, and if the larger position is not in the odd processor then the processors exchange objects. For the second comparison, processor pairs are formed along an axis by an even processor and the neighboring odd processor

with the smaller axis index: e.g., the processors with $i = 4$ (even) and $i = 3$ (odd) would form a pair. The coordinate values along the chosen axis for the two objects would be compared, but for this comparison, the smaller position should be in the odd processor. If the previous condition is false, a exchange of objects is made by the processors. A comparison set is performed for all axes, until no swapping occurs for any axis. This is just a multi-dimensional a bubble sort method. The bubble sort method is not very efficient for random data but for data only slightly unsorted, i.e., a few objects require exchanging, it is very efficient.

The satellites begin in MLG order, and as time progresses, the MLG order is perturbed to an increasing extent due to the random velocities of the satellites. Figures 6 and 7 show the relationship between MLG swaps and time steps. The average number of MLG swapping iterations per time step is plotted against the total number of time steps. The average number of MLG swapping iterations per time step approaches an asymptotic limit. The asymptotic nature reflects a maximum perturbation of the MLG from the initial orderliness of the satellites. Based on the figures, this asymptotic limit, which is to be expected, is a function of both step size and system size. The increase in step size results in larger positional changes, and the increase in system size results in more objects to sort, giving the $\ln N$ dependence.

Table 2 shows execution time ratios. The three columns give information about system size and VPR effects. Column 1 is data for the "small" system, $16 \times 16 \times 16$, and VPR = 1. Columns 2 and 3 are data for the "medium" system, $32 \times 32 \times 32$, and VPR = 2 and 4, respectfully. Row 1 is the program execution time per time step. The step size influences this ratio, but compared to system size and VPR, the influence is negligible. Row 2 is the execution time per MLG swapping iteration. Step size has no influence on this ratio. Rows 3 and 4 show the ratio of execution time divided by simulated time for step sizes of 10 and 20 seconds, respectively. Since step size has little effect on execution time, doubling the step size reduces the ratio by approximately 2.

Table 3 has execution time – simulated time ratios for the functions and operations of BEAST (see Fig. 1). The initialization section, which includes the housekeeping functions for BEAST, is not included in the table. This section of code requires about

10 to 15 seconds to run. It is independent of the scenario to be run and size of scenario, since this section is executed only once for the scenario. Compared to the other sections of BEAST, the initialization section uses the host alone to perform many functions. As an example, reading and writing the program's initialization files are performed by the host. The performance time is fixed by the number of steps performed by the host and will not increase with VPR. Therefore, the initialization section is not heavily dependent on VPR. Only execution time for the functions and operations within the loop will be depended on the simulated time and VPR. The simulated time and time step size determine the number of loops required by BEAST. The VPR will affect general communications and the amount of work for each physical processor. A VPR of 2 requires the routers and physical processors to perform the same operation twice, once for each virtual processor that the physical processor represents.

The total execution time summary at the table's bottom is important. For all scenario sizes and VPR, the BEAST code ran approximately three times or more faster than simulated time. This will allow more details to be added to BEAST. The principle execution time expenditure was in the report/tracker correlation function, which uses approximately half the time. General interprocessor communications are a major factor in the time used. Future research and program improvement should reduce the time used for tracking.

Table 3, which is for the latest BEAST code, and Table 2, which is for the initial trial code, are very similar both in arrangement and content. The execution time / simulated time ratios in Table 2 for step size = 10 s can be compared to the totals for the scenario generator and MLG near-neighbor sort in Table 3. While the operations performed are not the same, the functional purpose of updating the scenario and performing a MLG sort are the same. An increase in performance for the latest BEAST code is obvious. The latest code has reduced the axis communication required for the MLG sort. In addition Table 3 shows the same doubling effects of VPR on execution that Table 2 shows.

Execution Time Ratios for MLG Sort and System Update

Execution Time Ratios	System Size and VPR		
	16 × 16 × 16	32 × 32 × 32	
	VPR = 1	VPR = 2	VPR = 4
Execution Time per Time Step	.21	.55	.96
Execution Time per MLG Iteration	.05	.08	.13
Execution Time / Simulated Time			
step size = 10 s	.02	.051	.09
step size = 20 s	.012	.029	.055

Table 2: Execution Time Ratios

Performance of the Parallel Algorithms in BEAST Execution Time / Simulated Time Ratios

Functional Sections	Scenario Size and VPR		
	4096 Objects	32,768 Objects	
	VPR = 1	VPR = 2	VPR = 4
Scenario Generator:	0.0009	0.0017	0.0032
MLG Near Neighbor Sort:	0.0097	0.0405	0.0640
Sensor Report:	0.0114	0.0308	0.0558
Sensor Control:	0.0033	0.0074	0.0180
Multiple Sensor Fusion:	0.0072	0.0152	0.0250
Report/Track Correlation:	0.0481	0.0985	0.1830
Track Projection:	0.0001	0.0003	0.0005
Weapons Allocation:	0.0055	0.0103	0.0158
Total Execution Time Ratio:	0.0862	0.2047	0.3653

Table 3: Execution to Simulated Time Ratio

Appendix B: Tour

Normally, each processor has a single data set. Any operation requiring a second data set results in the communication between processors. The tour, mentioned in Section 4 and described in this appendix, uses near-communications along grid axes to access the second data set. The first step for a tour is the creation of a special data set within each processor called the touring data. Within each processor, the data to be compared and related data required for the procedure are copied into the touring data set. The original data are called the home data. As the touring data set moves from processor to processor by axes communications, the necessary calculations are performed on it and the processor's home data set. If an event occurs between the two data sets, a flag must be set in both data sets. The tour continues until the touring data returns home, and notifies the home data of any events or action to be taken.

This procedure can be pictured by the following description. Initially, a blue translucent cube exists and represents the original data set. A red translucent cube, that coincides with the blue cube, appears. The red cube represents the touring data set. The transfer of data from processor to processor appears as an offset of the red cube from the blue cube. This offset would change along one axis at a time. Finally, the red cube would return to its original position, as the tour data set did, and again coincides with the blue cube.

Now that a overview of the tour has been given, the details of the near-neighbor tour can be discussed. Since a number of processors require the touring data set, it is necessary to develop a path through each of these processors. This path should transfer the data set to all processors needing it, minimize the number of transfer operations and use the axis communication function. Figure 8 shows a drawing of a $3 \times 3 \times 3$ cube and the path used for its tour. The circles represent a central processor and the processors one index away. The solid line through the circles is the tour used. Each processor along the tour would be visited by the data from the central processor. For the circles without the line through them, their tours would result in their touring data sets visiting the central circle. This fact can be seen by choosing one of these circles as the central circle and following its tour. A similar tour was developed for a $5 \times 5 \times 5$

cube and is used for multiple sensor fusion.

The function for transferring data along an axis allows the choice of axis and direction. For the three-dimensional MLG being used, the axes are x, y or z and the direction is either positive or negative. The directions are stored in a file, consisting of a step number, an axis, positive or negative direction along the axis, and whether or not the calculation is to be performed. The last item is required for returning to the home processor and for visits to the same processor more than once. In both cases, the calculations are not required. The host computer stores the tour file and uses it to broadcast the transfer instructions and whether or not the calculations are to be performed for that step of the tour. For the $3 \times 3 \times 3$ cube, 14 steps are required, which results in 26 comparisons, for the home processor's data set. For the $5 \times 5 \times 5$ cube, 63 steps are required, which results in 124 comparisons for the home processor's data set.

Appendix C: Data Transfer Between Host and Processors

One key problem throughout the BEAST development on the CM has been the communications between the host and the processors. Testing of various communication methods resulted in decreasing execution time. In addition, this work shows the importance of using the CM processors compared to using the host alone. These tests were performed using the "small" MLG data structure and running on the CM's 4K section.

Initially, because only one grid could be used in the program, the SENSOR TANK was located on the host computer, resulting in a small section of serial code. Considerable effort would have been required to develop the SENSOR TANK code using the basic three-dimensional grid on the CM's processors. The cost for not having the programming effort was using host – processor communication methods. The two methods available are an array transfer or individual processor transfer. A third method, general broadcast, allows only communication from host to active processors (see Sec. 2). Three different algorithms using these communication methods were developed and were tested to reduce execution time.

The first algorithm used individual processor transfer, because only 64 data sets

required transferring. This transfer was accomplished by stepping through the 64 sensors using their identification numbers and transferring only the information needed for the SENSOR TANK. After the SENSOR TANK was sorted, each node within the SENSOR TANK represents a cube of processors within the REPORT TANK. Each cube requires information about the sensor at the corresponding node within the SENSOR TANK. Again, because of the small number, individual processor transfer was used, and only the data needed to access the sensor on the CM was transferred to a processor in the corner of the cube in the REPORT TANK. Then, axis communication using all the CM's processors transferred the sensor's data throughout the cube. All processors would then request additional information about the sensor assigned to them.

Since each processor within a cube in the REPORT TANK needs the same information, the host could broadcast that information to all processors within a cube. This second algorithm would transfer more data to the host, but hopefully, the speed of the general broadcast would result in a time saving. Surprisingly, this algorithm resulted in increased execution time. With study, the cause of the increased execution time was quickly discovered. One of the slowest communication methods is individual processor communication to the host. The general broadcast is fast because it is continuously being divided by the routers and reaching more and more processors. The individual processor communication to the host must reverse the path. Therefore, it will be slowed by the numerous reduction steps. The present C* instruction set resulted in one variable at a time being transferred from the processor to the host and the host to processor. The conclusion from this modification was to reduce data transfer from individual processor to the host.

The next algorithm combined the first and second algorithms. Only the information required for the SENSOR TANK was transferred to the host, and only the information required by the processors to access the sensor's data were broadcasted to the cube in the REPORT TANK. The processors, then, requested the additional sensor information. This algorithm resulted in a slight decrease in execution time in comparison to the initial algorithm. Further study revealed that some of the additional sensor information could be calculated. Additions made to the code and the NRL C* library perform these calculations, resulting in another slight decrease in execution time.

All the previous algorithms used the host computer to store and to sort the SENSOR TANK. These algorithms required a section of serial code to be run on the host resulting in the CM's processors being idle. A major advance was made to the code and NRL C* Library when multiple processor grids were allowed in a program. The present algorithm presented in Section 5 uses the multiple grids. In comparison to the initial algorithm, the present algorithm reduced total execution time by approximately 10% for the BEAST code without the weapon allocation or tracker-correlation sections.

This discussion and work lead to an "important" conclusion. All CM processors should be used to perform a task! As always, there will be exceptions to the rules, but only testing will find them. The gain in speed caused by moving the SENSOR TANK from the host to the processors easily overcame the loss in speed caused by general communications. Similar statements can generally be made any time more processors can be used for a task or procedure.

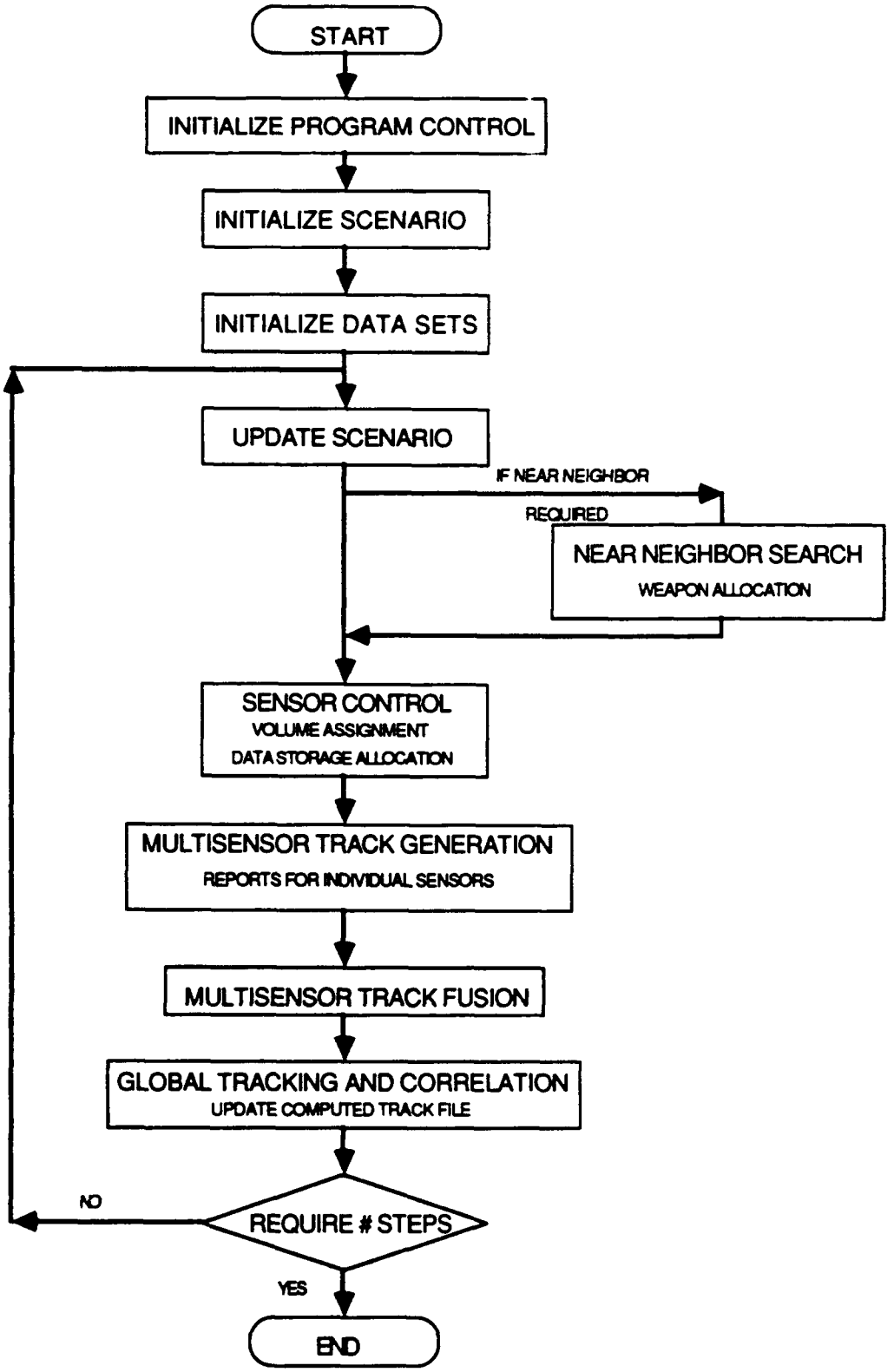
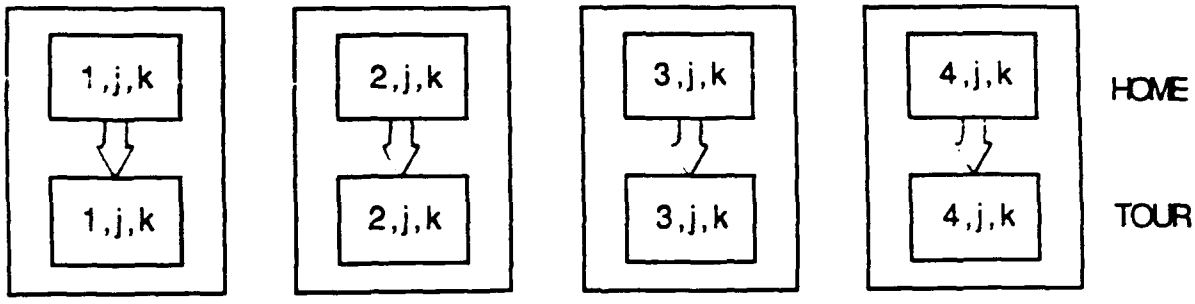
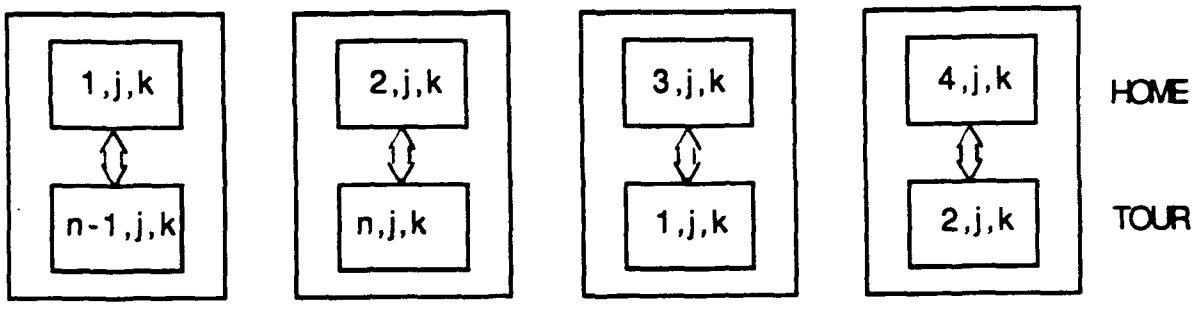


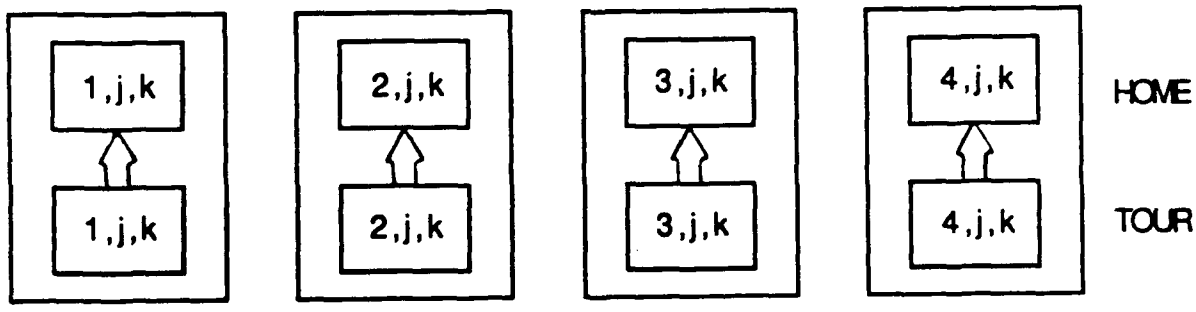
Figure 1: Block Diagram of the BEAST code



Part A Starting Position



Part B During Tour



Part C After Tour

Figure 2: Touring Data visiting Processors

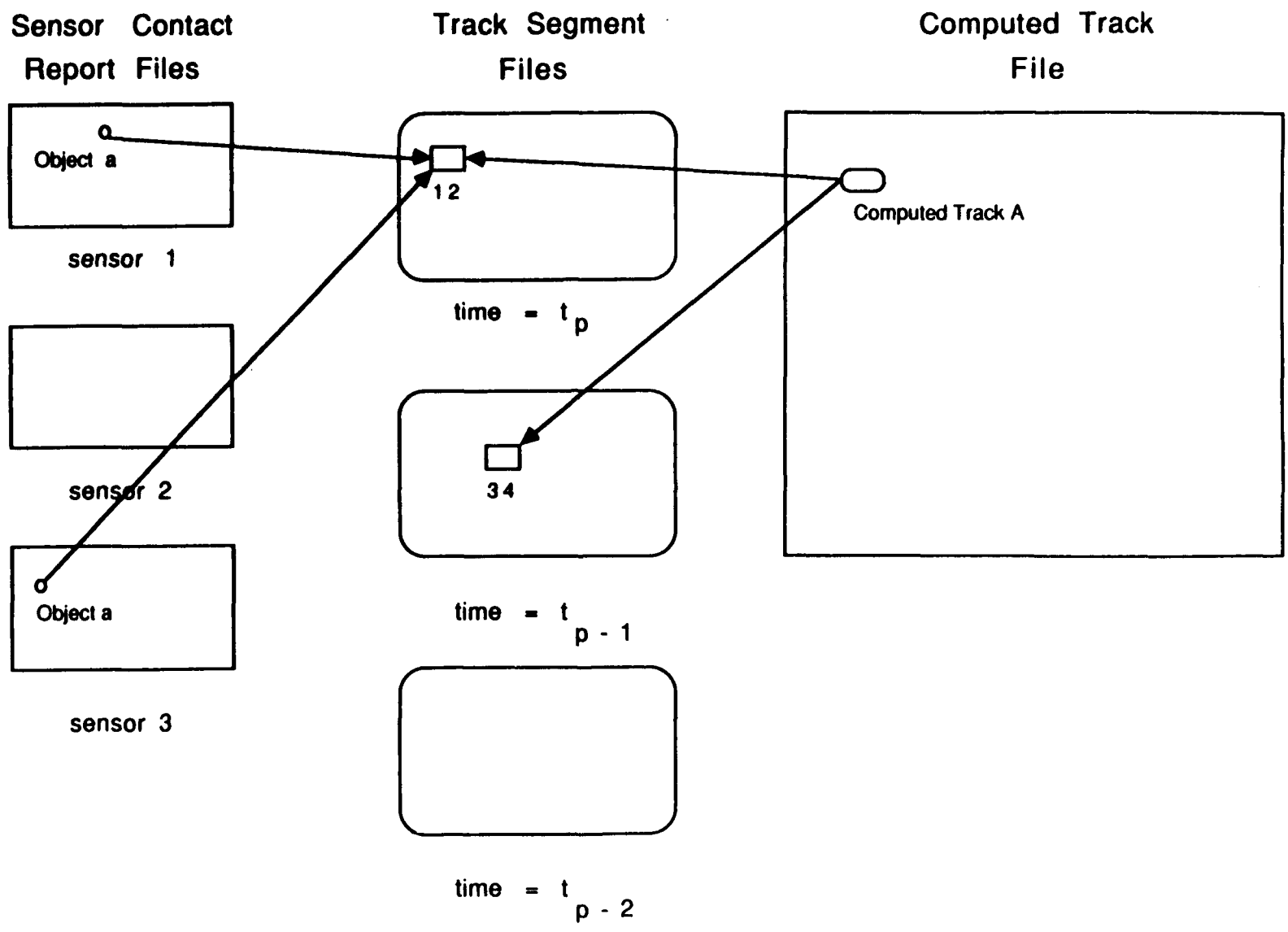


Figure 3: The sensor contact reports, track segments and track relations

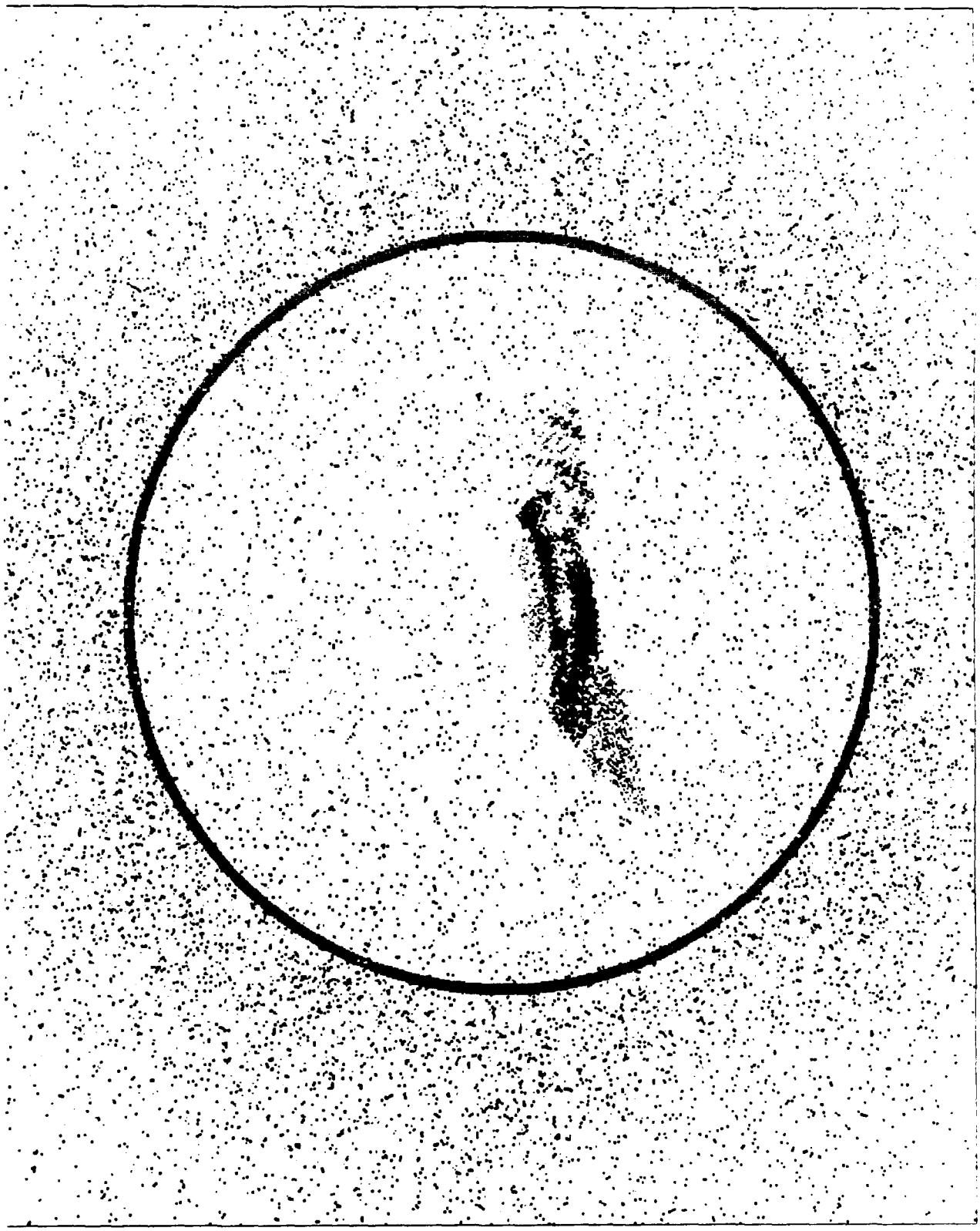


Figure 4: 32,768 Node Truth Tank

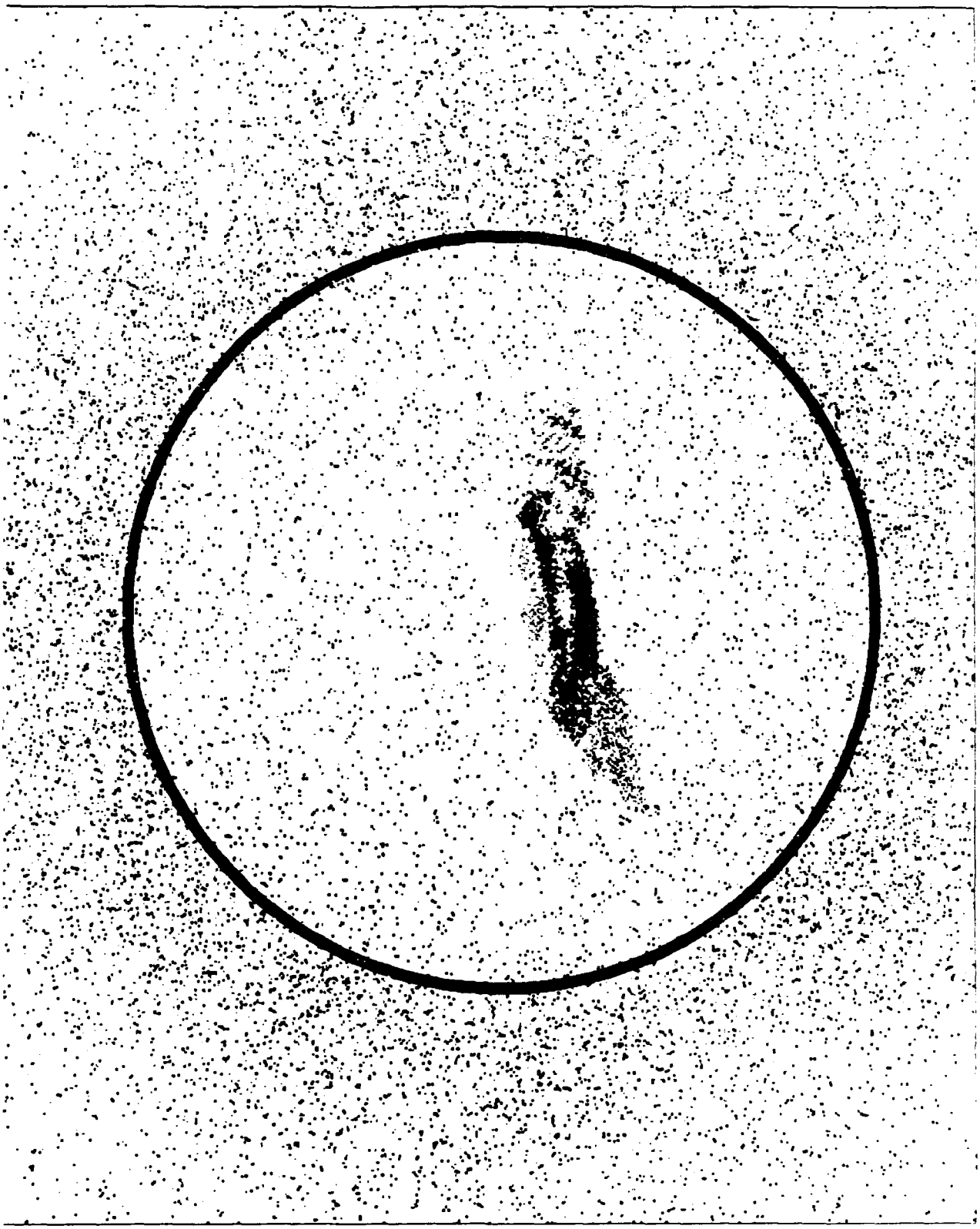


Figure 5: 32,768 Node Sensor Report Tank MLG

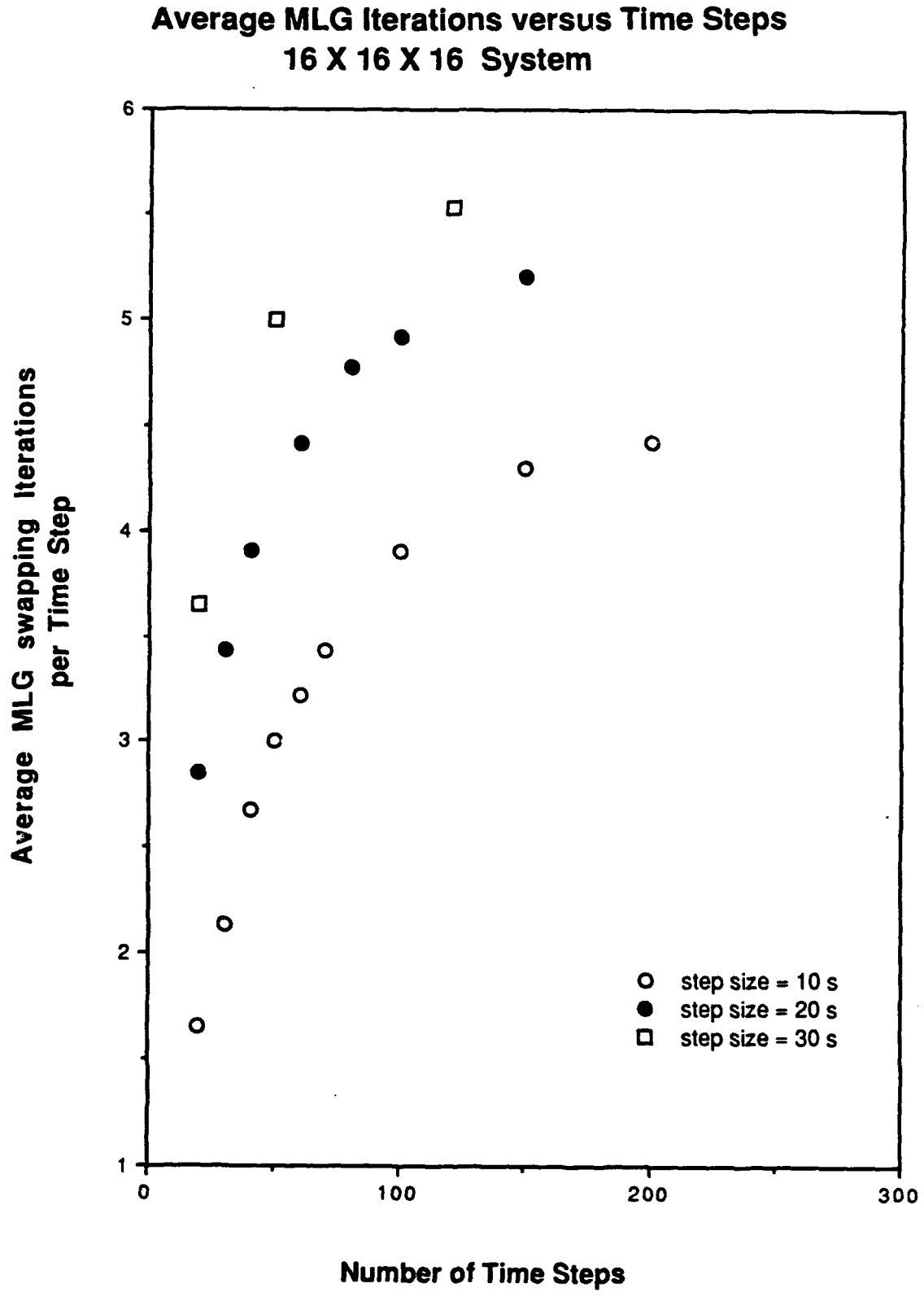


Figure 6: Average MLG Swaps versus Time Step for 16 x 16 x 16 System

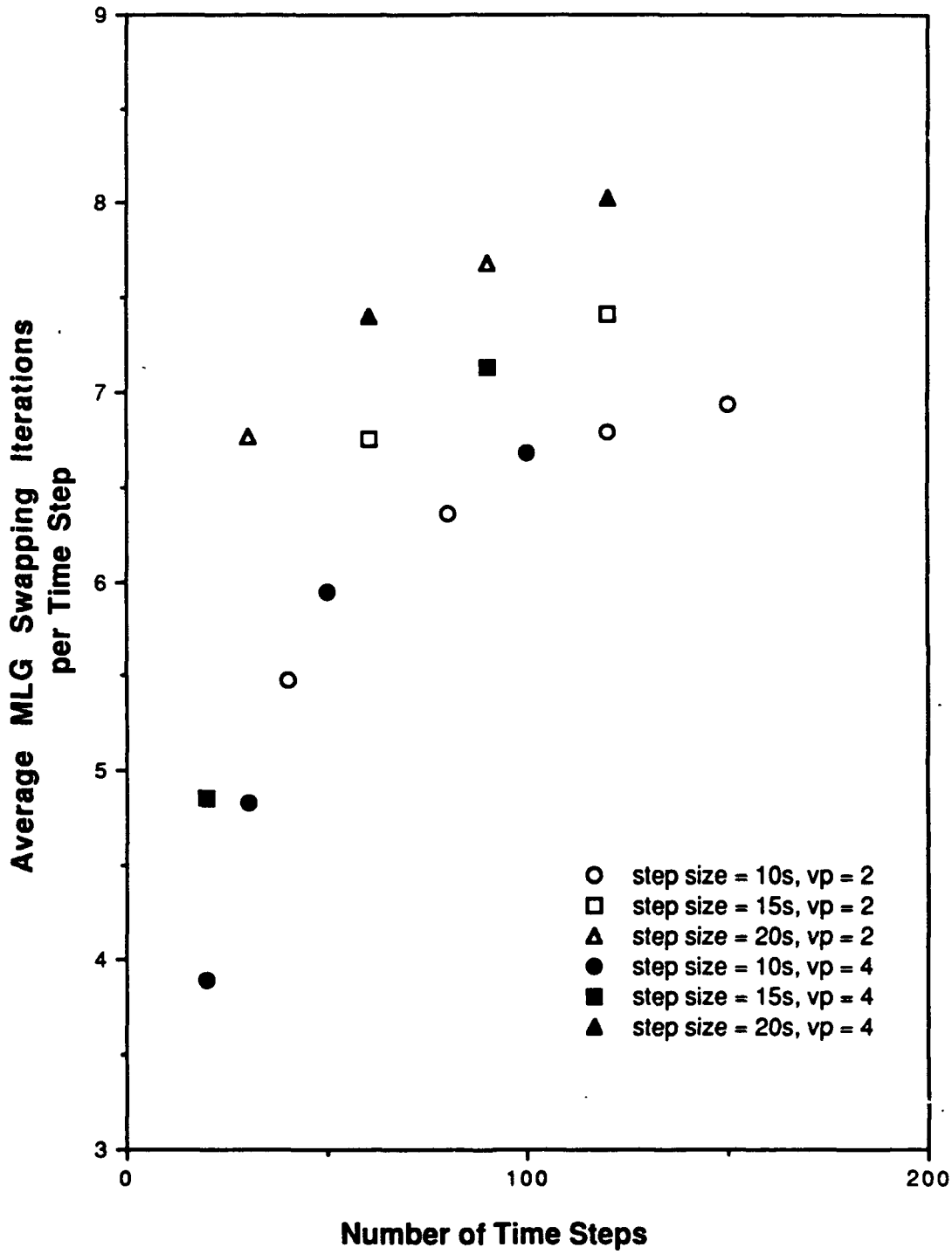
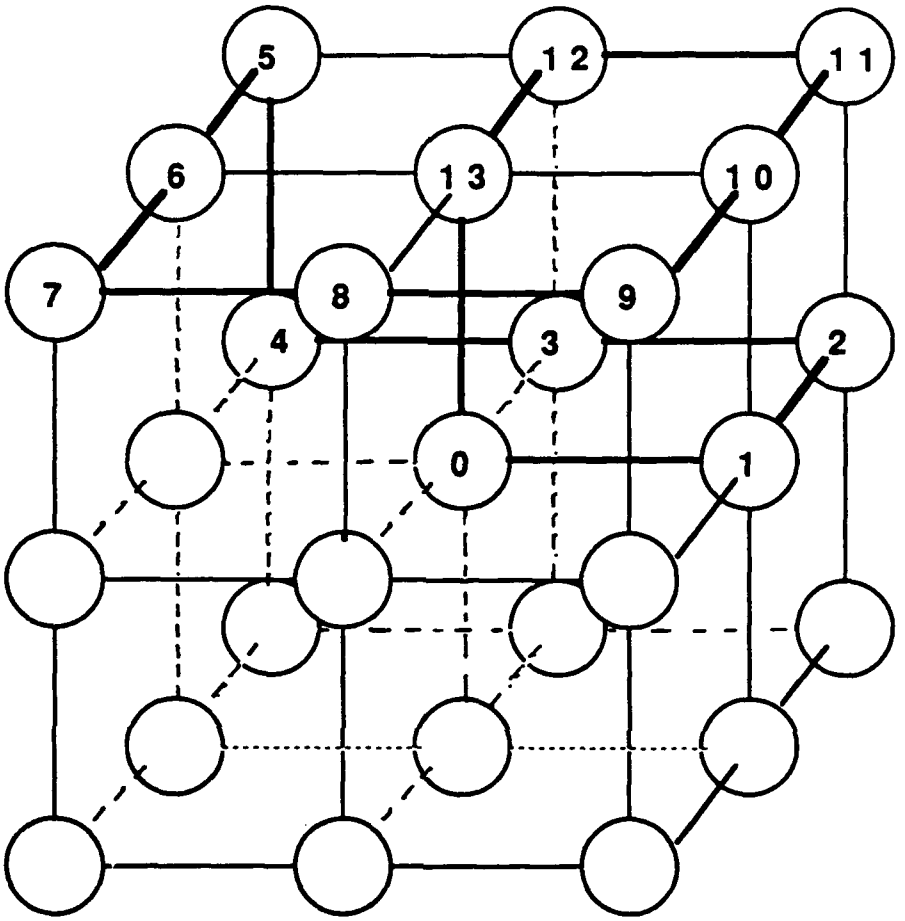


Figure 7: Average MLG Swaps versus Time Step for 32 x 32 x 32 System



3 X 3 X 3 TOUR

Figure 8: Tour of processors one index away